POWER TOOLS

What's GNU, Part Seven

By Jerry Peek

A lot of utilities have changed since the early days of *Unix*. This month, let's take one last look at new features added to a few of the most common Linux utilities, including *diff*, *wc*, *du*, *date*, *touch*, and *sed*.

What's Different About diff

GNU *diff version 2.8.1* has more than forty options. (The *Seventh Edition diff* had four.) Covering all forty is impractical, so let's focus on the customizable output formats of GNU *diff*.

The Seventh Edition *diff* had its default output format and the `-e` and `-f` formats. Later, the *context* (`-c`) format was added, then came the unified (`-u`) format. Now *diff* has the `-D` *NAME* option to merge two C language files into a single file with the preprocessor directives `#ifdef` *NAME* and `#ifndef` *NAME*. Modern *diff* also has a series of options such as `--line-format=`*fff*, where *fff* is a `printf()`-like string, that controls how *diff* formats difference listings. The latter series of options lets you produce almost any type of output.

Listing One shows two short files with the normal output format, the unified format, and a custom format using the line-format options.

The shell script */tmp/differ* shown in *Listing One* uses a custom listing format. The options `--old-line-format` and `--new-line-format` each have a multi-line argument: the blank lines in the arguments cause the blank lines on the output, and the `%L` is replaced by the line of text. The empty format for the option `--unchanged-line-format` means that unchanged lines aren't emitted. Although you can type these multiline formats on a command line, it's probably easier to write a little shell script.

What's Changed in wc

The "word count" utility, *wc*, has always counted lines, words, and characters in a file. The original version assumed that a character was a single byte, but newer versions have the `-m` option to count (possibly) multi-byte characters.

Older versions padded the counts with multiple space characters:

```
$ wc somefile
       26      390     2706 somefile
```

The GNU version uses less space unless it's reading standard input:

```
$ wc somefile
  26  390 2706 somefile
$ echo -e "just a\ntest" | wc
       2       3      12
```

LISTING ONE: A sampling of different *diff* listing formats

```
$ diff old new
1a2
> line 1A
3d3
< line 3
5a6
> line 6

$ diff -u old new
—- old 2006-01-16 14:35:27 -0700
+++ new 2006-01-16 14:35:52 -0700
@@ -1,5 +1,6 @@
  line 1
+line 1A
  line 2
-line 3
  line 4
  line 5
+line 6

$ cat /tmp/differ
#!/bin/sh
diff \
     —old-line-format '
ONLY IN THE OLD FILE:

%L' \
     —new-line-format '
ONLY IN THE NEW FILE:

%L' \
     —unchanged-line-format='' \
   old new

$ /tmp/differ

ONLY IN THE NEW FILE:

line 1A

ONLY IN THE OLD FILE:

line 3

ONLY IN THE NEW FILE:

line 6
```

The new version of *wc* also has the `-L` and `--max-line-length` options to count the longest input line (not including the newline):

```
$ wc -L /usr/share/dict/words
23
$ egrep '.{23}' /usr/share/dict/words
electroencephalograph's
```

The longest word in the test system's dictionary file has 23 characters, and it's "electroencephalograph's." `wc -L` doesn't output the filename, unless you also use another option, such as `-w`.

Disparities in du

The original *du* generally gave results as the number of 512-byte blocks. Later, `du -k` counted in 1K-byte blocks. The GNU version has the options `-B` and `--block-size` and the `BLOCK_SIZE` environment variable to let you choose your own scale. The block size settings also apply to GNU *df* and *ls*. For example, *Listing Two* first uses the default size, then 512-byte blocks.

A series of abbreviations (listed in the *info* pages node "Block size") lets you choose units like `M` (Megabytes, 2^{20}) or `MB` (Megabytes, 10^6). You can also add a numeric multiplier. For instance, to show the size of */usr/bin* in kilobyte (2^{10}), megabyte (2^{20}), and 10-megabyte ($10 * 2^{20}$) units, you can type the following commands, respectively:

```
$ du /usr/bin
173948   /usr/bin
$ du -B M /usr/bin
170M     /usr/bin
$ du -B 10M /usr/bin
17       /usr/bin
```

LISTING TWO: Changing the default block size for GNU *du*, *df*, and *ls*.

```
$ ls -sF /usr/share/dict
total 892
892 american-english     0 words@
$ du /usr/share/dict
896      /usr/share/dict
$ export BLOCK_SIZE=512
$ ls -sF /usr/share/dict
total 1784
1784 american-english      0 words@
$ du /usr/share/dict
1792     /usr/share/dict
```

The `--apparent-size` option makes `du` work like `wc -c` or `ls -l`: it counts the number of bytes saved in the file instead of the disk usage (total size of the file's disk blocks). For instance, *Listing Three* makes a one-character file named *small* and a huge but sparse file named *big*. Plain *du* shows that *small* takes an entire block (4,096 bytes) to store, but `du --apparent-size` shows that it only holds one byte. The *big* file contains nothing, but holds a lot of disk space hostage.

GNU diff 2.8.1 has more than forty options. (The *Seventh Edition diff* had four.)

The *du* option `--exclude` lets you give a shell wildcard pattern of files that shouldn't be counted. The related option `--exclude-from` is for a filename with multiple patterns, one per line; a filename of `-` (a hyphen) reads the list from standard input.

So, to skip all directories and files whose names start with an uppercase English letter:

```
$ du --exclude='[A-Z]*'
1234     .
```

To skip all files ending with *.doc*, *.ppt*, and *.sxc*, use *echo* to create a newline-separated list. The shell reads every pattern until you type the closing quote. (This is harder to do if you use a C shell like *tcsh*.)

```
$ echo '*.doc
> *.ppt
> *.sxc' | du --exclude-from=-
604      .
```

Some of the other new options control whether symbolic links are dereferenced (showing the disk space used by what the link points to instead of the space used by the link itself) or not, whether to cross filesystems, and whether to show the size of each directory separately without including subdirectory sizes.

LISTING THREE: Finding the sizes of small and big files with `du --apparent-size`

```
$ echo -n x > small

$ du -B 1 small
4096     small

$ du -B 1 --apparent-size small
1        small

$ dd bs=1 seek=2000000000000 of=big < /dev/null
0+0 records in
0+0 records out
0 bytes transferred in 0.000115 seconds (0 bytes/sec)

$ du -B GB --apparent-size big
2000GB   big
$ du big
0        big

$ ls -l
total 4
-rw-r--r--  1 jpeek users 2000000000000 2006-01-16 18:29 big
-rw-r--r--  1 jpeek users             1 2006-01-16 18:29 small
```

Updates to date

date used to simply print the date in a standard format (although administrators could also use *date* to set the system clock). Later, you could set the output format with `printf()`-like specifications that start with a plus sign, like this:

```
$ date '+Today is %A, %B %e.'
Today is Monday, January 16.
```

GNU *date* also accepts a date string after the `-d` or `--date` option to specify what time to use. For instance, if you're testing a program, you could make it print the time string you would get running the program at 6 PM yesterday. You can give relative times like `-1 hour`. It's also handy for date conversions — for instance, finding what day of the week yesterday was (where 0 represents Sunday):

```
$ date -d '6 pm yesterday'
Sun Jan 15 18:00:00 MST 2006
$ date -d yesterday '+%w'
0
```

The default *date* output is locale-dependent. You can also choose a RFC-2822 (email)-compliant format…

```
$ date -R
Mon, 16 Jan 2006 19:07:57 -0700
```

… or an ISO 8601 format including only the date; the date and hours; the date, hours and minutes; or date, hours, minutes, and seconds. The latter three are followed by the timezone:

```
$ date -Idate
2006-01-16
$ date -Ihours
2006-01-16T19-0700
$ date -Iminutes
```

```
2006-01-16T19:10-0700
$ date -Iseconds
2006-01-16T19:10:41-0700
```

Or how about Coordinated Universal (Greenwich) time?

```
$ date -u
Tue Jan 17 02:11:05 UTC 2006
```

Finally, you can display the last-modification time of a file in any format *date* can handle:

```
$ touch -t 200901020304 ts
$ ls -l ts
-rw-r—r— ... 2009-01-02 03:04 ts
$ date -r ts
Fri Jan  2 03:04:00 MST 2009
$ date -r ts '+%w'
5
```

Tweaks to touch

The original *touch* utility would create an empty file or change the last-modified time of an existing file to "now." Later, the `-t` option let you specify any modification time in the past or future. The GNU version of *touch* has the same `-d` option as *date*, which lets you describe a time in words like month names, "yesterday," relative times, and more.

Even more handy is the new `-r` or `--reference` option that lets you "copy" the timestamp from a reference file — that is, to make your file's timestamp the same as another file. You can also use `-d` with the reference option to set a relative time. For example, to set the last-modification time of *afile* to one minute before *bfile:*

```
$ ls -l bfile
-rw-r—r— ... 2006-01-16 19:40 bfile
$ touch -d'-1 minute' -r bfile afile
$ ls -l afile
-rw-r—r— ... 2006-01-16 19:39 afile
```

The `-a` option changes a file's last-access time instead of its last-modification time.

Saves in sed

sed is a "stream editor," or an editor designed to read text from files or standard input, change contents, and write the result to standard output. But it's often used to edit a file by redirecting output to a temporary file and using that to replace the original file. (*sed* uses commands like

ed — which is designed for editing files, not standard input — but *sed* has loop, branch, and test commands that *ed* doesn't.)

The new options `-i` or `--in-place` direct *sed* to edit files in place. If you give a suffix after the option, *sed* makes a backup before editing.

For instance, to edit the files *afile* and *bfile* in place, "copying" them to *afile.bak* and *bfile.bak* before editing, type:

```
$ sed -i.bak 's/old/new/' [ab]file
```

One word of caution: If you use the `-i` option, the backup suffix must come immediately after the option with *no space between*, just as shown here.

`touch -r` lets you "copy" the timestamp from a reference file — that is, to make your file's timestamp the same as another file

If you use one or more asterisk (`*`) characters in the argument to `-i` or `--in-place`, each asterisk is replaced by the current filename. So, to make the backups in a directory named *bk*, use:

```
$ Bsed -i'bk/*.bak' 's/old/new/' [ab]file
```

The backup of *afile* is saved as *bk/afile.bak*, and *bfile* is saved as *bk/bfile.bak*.

The `-i` option also sets the new `-s` option, which treats each file separately instead of (as the original *sed* did) one long stream. Line numbers reset in each new file, `$` refers to the last line of each file, and so on.

Another handy new option is `-u` for minimal buffering. It's useful when you're trying to edit the output of a program like `tail -f` where data may come slowly and you want to see the results as soon as possible.

That's All, Folks…

There are many, many more GNU utilities, but it's time for this column to move on to a new topic. Check the documentation for any utilities you use. The *info* version is usually more complete than the *man* page to see what else is GNU.

Jerry Peek is a freelance writer and instructor who has used Unix and Linux for 25 years. He's happy to hear from readers; see http://www.jpeek.com/contact.html.